2Б

УДК 681.3

М.К. Буза
Белорусский государственный университет, г. Минск, Беларусь
Беларусь, г. Минск, пр-т Независимости, 4, *bouza@bsu.by*

Алгоритмическая и программная реализация DSM-памяти

M.K. Bouza
Belarusian State University
Pr. Nezavisimosti 4, Minsk, Belorussia, bouza@bsu.by

Programming Support DSM-Memory

М.К. Буза
Білоруський державний університет, м. Мінськ, Білорусь
Білорусь, м. Мінськ, пр-т Незалежності, 4, *bouza@bsu.by*

Алгоритмічна і програмна реалізація DSM-пам'яті

Предложена и апробирована программная система DSM-памяти на основе разделяемых объектов. Она обеспечивает передачу данных в одном адресном пространстве. При реализации системы использована технология XSTM для репликации объектов между процессами.
Ключевые слова: распределенная память, репликация, тестирование, когерентность.

Programming system DSM – memory on the base of distributed objects is suggested. System supported data transfer in common address. Technology XSTM was used for replication objects between processors.
Key words: shared memory, replication, testing, coherence.

Запропонована і апробована програмна система DSM- пам'яті на основі розділюваних об'єктів. Вона забезпечує передачу даних в одному адресному просторі. При реалізації системи використана технологія XSTM для реплікації об'єктів між процесами.
Ключові слова: розподілена пам'ять, реплікація, тестування, когерентність.

Введение

Для решения сложных научных и инженерных задач требуются высокопроизводительные масштабируемые вычислительные системы параллельного действия, которые сегодня создаются на основе кластерных технологий. Однако эффективность использования таких архитектур существенно зависит от наличия моделей и средств проектирования распределенных приложений [1], [2]. Одним из требований к таким моделям и средствам сегодня является степень масштабируемости.

Кластерные системы, построенные на базе высокоскоростных сетевых технологий (*Fast Ethernet, Gigabit Ethernet*) имеют большую латентность и меньшую производительность, чем системы, построенные на специализированных интерфейсах (*SCI, Myrinet, Infiniband*) [3].

Так, например, в кластерах «СКИФ» используются на первом уровне сети *Fast Ethernet* или *Gigabit Ethernet* для функции управления, на втором, построенном на базе интерфейсов *SCI*, *Infiniband* или *Myrinet*, обеспечивается высокоскоростной транспорт данных между узлами кластера. Обычно подобные интерфейсы позволяют создавать связи между узлами с гиперкубической топологией.

Однако для ряда классов задач с последовательно-параллельными вычислениями с небольшой долей операций обмена данными между параллельными ветвями кластерные системы на базе высокоскоростных сетевых технологий выигрывают по соотношению стоимость / производительность.

Классические методы разработки распределенных приложений базируются на модели передачи сообщений, поддерживаемой коммуникационной библиотекой *MPI* (*Message Passing Interface)* [4]. Доступ к разделяемым данным в этой модели обеспечивается посредством явных операций посылки и приема сообщений, что сильно усложняет программирование, отладку программ и применение для масштабируемых приложений, которые не должны зависеть от количества процессоров.

Более привлекательной моделью для построения распределенных масштабируемых приложений для кластерных систем является модель общей распределенной памяти DSM (*Distributed Shared Memory)* [2].

DSM – виртуальное адресное пространство, разделяемое всеми узлами (процессорами) распределенной системы. Использование DSM-модели при разработке распределенных приложений позволяет уделять основное внимание алгоритмическим вопросам, а не разделению данных и коммуникации. Создав абстракцию общей памяти, передача данных будет осуществляться в одном адресном пространстве.

В связи с этим актуальной становится **задача** разработки программной системы обеспечения консистентности объектов и универсального доступа к ним.

Цель данной работы – использование DSM-модели при создании абстракции общей памяти для реализации передачи данных в одном адресном пространстве.

1 Исследование алгоритмов когерентности памяти в распределенных системах

DSM-память может быть реализована на различных уровнях: аппаратном и программном. Примером аппаратной реализации являются системы, построенные по технологии NUMA (Nou Uniform Memory Access) [5]. Наряду с доступом к собственной локальной памяти все узлы с помощью специальной логики имеют доступ к пространству оперативной памяти других узлов.

В данной работе предложена программная реализация DSM-памяти с использованием технологии репликации объектов.

1.1 Основные алгоритмы реализации DSM

Главная цель реализации DSM-модели – повышение производительности системы за счет обеспечения доступа к разделяемым данным одновременно на нескольких узлах.

Алгоритм с центральным сервером. Все разделяемые данные поддерживает центральный сервер. Он возвращает данные клиентам по их запросам на чтение, запись, корректирует данные и посылает клиентам в ответ квитанции. Клиенты могут использовать тайм-аут для посылки повторных запросов при отсутствии ответа сервера. Дубликаты запросов на запись могут распознаваться путем нумерации запросов. Если несколько повторных обращений к серверу остались без ответа, приложение получит отрицательный код ответа. Чтобы избежать этого, разделяемые данные могут быть распределены между несколькими серверами. Алгоритм прост в реализации, но сервер может стать узким местом.

Миграционный алгоритм. Он обеспечивает перемещение данных в точку их востребования в текущий момент времени. Это позволяет последовательные обращения

к данным осуществлять локально. Миграционный алгоритм позволяет обращаться к одному элементу данных в любой момент времени только одному узлу.

Обычно мигрирует целиком страницы или блоки данных, а не запрашиваемые единицы данных для снижения стоимости миграции. Однако такой подход приводит к трэшингу, когда страницы очень часто мигрируют между узлами при малом количестве обслуживаемых запросов.

Алгоритм размножения для чтения. Данный алгоритм расширяет миграционный алгоритм механизмом размножения блоков данных, позволяя либо многим узлам иметь возможность одновременного доступа по чтению, либо одному узлу иметь возможность читать и писать данные. Производительность повышается за счет возможности одновременного доступа по чтению, но запись требует серьезных затрат для уничтожения всех устаревших копий блока данных или их коррекции.

Алгоритм полного размножения. Алгоритм позволяет многим узлам иметь одновременный доступ к разделяемым данным на чтение и запись (протокол многих читателей и многих писателей). Поскольку многие узлы могут писать данные параллельно, требуется для поддержания согласованности данных контролировать доступ к ним.

Одним из способов обеспечения консистентности данных является использование специального процесса для упорядочивания модификаций памяти. Все узлы, желающие модифицировать разделяемые данные, должны посылать свои модификации этому процессу. Он будет присваивать каждой модификации очередной номер и рассылать его широковещательно вместе с модификацией всем узлам, имеющим копию модифицируемого блока данных.

Для повышения эффективности рассмотренных алгоритмов необходимо изменить семантику обращений к памяти.

1.2 Модели консистентности DSM-памяти

Особую значимость для систем распределенной памяти имеют модели консистентности, которые представляют собой некоторое соглашение между программами и памятью, в котором указывается, что при соблюдении программами определенных правил работа модулей памяти будет корректной [2]. Ниже приведены модели консистентности, используемые в системах с распределенной памятью. Различия между ними определяются ограничениями моделей, простотой реализации и способами программирования:

Строгая консистентность – модель консистентности, удовлетворяющая условию: операция чтения ячейки памяти с адресом x должна возвращать значение, записанное самой последней операцией записи с адресом x.

Строгая консистентность представляет собой идеальную модель для программирования, но ее, к сожалению, невозможно реализовать для распределенных систем. Однако практический опыт показывает, что в некоторых случаях можно обходиться и более «слабыми» моделями.

Последовательная консистентность – результат выполнения должен быть тот же, как если бы операторы всех процессоров выполнялись в некоторой последовательности, определяемой программой этого процессора.

Это определение означает, что при параллельном выполнении все процессы должны «видеть» одну и ту же последовательность записей в память.

Последовательная консистентность только точно гарантирует, что все процессы знают последовательность всех записей в память. Результат повторного выполнения параллельной программы в системе с последовательной консистентностью (как, впро-

чем, и при строгой консистентности) может не совпадать с результатом предыдущего выполнения этой же программы, если в программе нет регулирования операций доступа к памяти с помощью механизмов синхронизации.

Причинная консистентность представляет собой более слабую модель по сравнению с последовательной моделью, поскольку в ней не всегда требуется, чтобы все процессы видели одну и ту же последовательность записей в память, а проводится различие между потенциально зависимыми и независимыми операциями записи. При этом последовательность операций записи, которые потенциально причинно зависимы, должна наблюдаться всеми процессами системы одинаково, параллельные операции записи могут наблюдаться разными узлами в разном порядке.

Система DSM может осуществить такую зависимость посредством нумерации всех записей на каждом процессоре, распространения этих номеров по всем процессорам вместе с модифицируемыми данными и задержки любой модификации на любом процессоре до тех пор, пока он не получит все те модификации, о которых известно процессору – автору задерживаемой модификации.

Слабая консистентность основана на выделении среди переменных специальных синхронизационных переменных и описывается следующим: доступ к синхронизационным переменным определяется моделью последовательной консистентности и запрещен (задерживается), пока не выполнены все предыдущие операции записи или пока не выполнены все предыдущие обращения к синхронизационным переменным.

Консистентность по выходу – введены специальные функции обращения к синхронизационным переменным: *acquire* – захват синхронизационной переменной, информирует систему о входе в критическую секцию; *release* – освобождение синхронизационной переменной, определяет завершение критической секции. Захват и освобождение используется для организации доступа не ко всем общим переменным, а только к тем, которые защищаются данной синхронизационной переменной.

Консистентность по входу – пример модели консистентности, которая ориентирована на использование критических секций. Как и в предыдущей модели, эта модель консистентности требует от программистов (или компиляторов) использование механизма захвата / освобождения для выполнения критических секций. Однако здесь требуется, чтобы каждая общая переменная была связана с некоторой синхронизационной переменной (типа семафора или события), при этом если доступ к элементам массива, или различным отдельным переменным, может производиться независимо (параллельно), то эти элементы массива (общие переменные) должны быть связаны с разными синхронизационными переменными.

2 Модель абстракции распределенного объекта

Распределенная система должна предоставлять приложениям удобный интерфейс ко всем ресурсам. С этой целью предлагается реализация DSM-памяти как образ единой системы [6]. Это означает, что для прикладного программного обеспечения распределенная вычислительная система выглядит как централизованная. Данное свойство позволяет разработчику абстрагироваться от физического расположения ресурсов, с которыми взаимодействует приложение и сосредоточиться на функциональности, которую они предоставляют. Это можно представить, как показано на рис. 1.

Реализация образа единой системы основана на абстракции распределенного объекта. Каждый объект предоставляет набор четко определенных интерфейсов, которые могут вызываться другими объектами. Объекты глобально, единообразно доступны посредством своих интерфейсов из всех узлов кластерной системы.

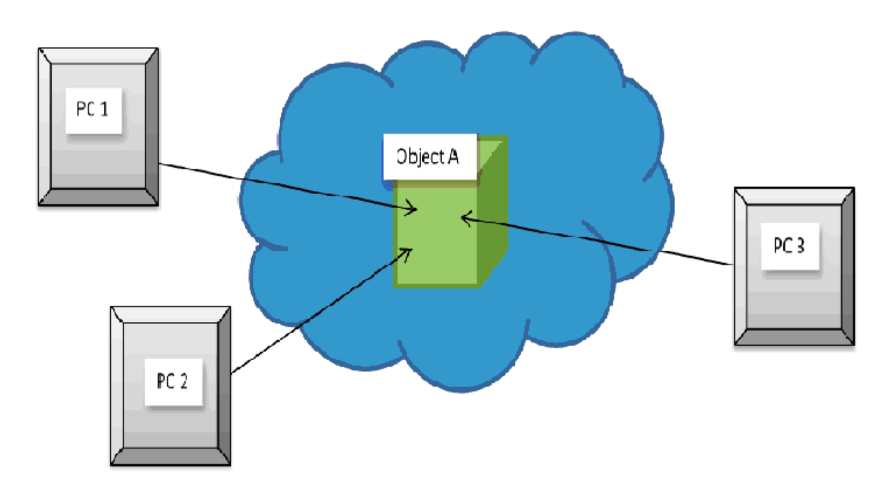


Рисунок 1 – Модель распределенного объекта

Ниже предлагается использовать репликацию для обеспечения эффективности и надежности доступа к распределенным объектам.

Распределенные системы состоят из набора взаимодействующих компонентов, расположенных в различных узлах сети. Поскольку операции, выполняемые в каждом узле, часто зависят от команд и данных, получаемых от удаленных компонентов, задержки, возникающие при распределенном взаимодействии, в конечном итоге снижают эффективность функционирования всей системы.

Для преодоления данного эффекта, в распределенных приложениях можно использовать такие приемы, как замена удаленного взаимодействия локальными операциями и вынесение удаленного взаимодействия за рамки критических путей выполнения программы. Замена удаленного взаимодействия локальным предполагает, что состояние объекта-сервера кэшируется в узлах, где находятся клиенты. В этом случае операции чтения могут выполняться локально над кэшированной копией объекта. Операции записи также иногда могут выполняться локально с последующей доставкой пакета изменений серверу. Метод вынесения удаленного взаимодействия за рамки критических путей направлен на то, чтобы сократить время, затрачиваемое основными вычислительными потоками на ожидание получения удаленного сообщения. Для этого используются вспомогательные потоки, спекулятивным образом получающие данные, требуемые для выполнения основных вычислений.

Обобщением указанных подходов является репликация объектов. Состояние распределенного объекта может быть полностью или частично доступно локально в каждом узле, где данный объект используется. Состояние объекта синхронизируется (реплицируется) между узлами. Каждый вызов метода объекта вначале обрабатывается его репликой, находящейся в том узле, где производится данный вызов. Взаимодействие с удаленными репликами привлекается только в тех случаях, когда этого требует протокол репликации, например, когда необходимо получить недостающую часть состояния объекта.

Таким образом, распределенное взаимодействие перемещается вовнутрь распределенного объекта. Следовательно, эффективность доступа к объекту определяется эффективностью стратегии репликации. Очевидно, что не существует стратегии репликации, одинаково эффективной для всех типов объектов. Поэтому необходимо предоставить механизмы для построения реплицированных объектов. При этом для каждого класса объектов может использоваться алгоритм репликации, обеспечивающий максимально эффективный доступ с учетом семантики объекта. Такой алгоритм может быть выбран из набора существующих стратегий репликации либо разработан специально для данного класса объектов.

3 Репликация объектов

Основные требования, которые предъявляются к системе репликации, состоят в том, что репликация должна быть прозрачной – обращение клиента к сервису не должно отличаться в случае использования сервисом данных реплики и реплицированные данные должны быть целостными. В общем виде задача репликации довольно сложна. Отметим некоторые частные случаи, встречающиеся в практических приложениях:

Пассивная (*primary – backup*) репликация. Изменение данных допускается только на одном узле (мастер – копия), а на всех других узлах данные доступны только для чтения. Она применима, например, для резервного копирования данных в системах, где операций изменения данных сравнительно мало по отношению к операциям извлечения больших объемов данных.

Активная репликация. Это ситуация, когда несколько узлов могут претендовать на изменение объекта.

В данной работе, которая ориентирована на разработку систем проектирования распределенных приложений для задач моделирования и вычислительного эксперимента, акцент делается на транзакционные механизмы и активную репликацию с блокировками, которые имеют более универсальный характер.

В работе предложена собственная методика репликации объектов. Методика основана на активной репликации с блокировкой. Главным ее достоинством является поддержка частичной блокировки. Проведено исследование и экспериментальная реализация этой методики для простых объектов.

При реализации системы использована технология XSTM (Extended Software Transactional Memory) для репликации объектов между процессами. В настоящее время при разработке распределенных приложений предъявляются требования их платформенной независимости. В связи с этим при проектировании приложений используется платформенно-независимый язык Java. Он является отличным выбором как для проведения исследований в области высокопроизводительных вычислений, так и для использования его при разработке промышленно эксплуатируемых приложений для кластерных суперкомпьютерных систем и грид-сетей. В данной предметной области Java практически не уступает в производительности языкам C, C++, но при этом значительно опережает их в удобстве и скорости разработок.

В связи с этим программная поддержка созданной DSM-системы реализована с использованием технологий .NET и Java.

При тестировании созданной DSM-системы было разработано распределенное приложение решения СЛАУ.

4 Тестирование системы rDSM

Разработанное распределенное приложение решения СЛАУ демонстрирует эффективность использования DSM-памяти. В работе это приложение используется также для тестирования replication DSM (rDSM) системы.

Основным параметром теста является N – размерность матрицы системы линейных уравнений. Результатом теста является общее время T_{exec_m} решения задачи размерности N в зависимости от числа процессоров m и коэффициент ускорения вычислений $\rho = T_{exec_m} / T_{exec_1}$, где T_{exec_1} – время решения задачи на одном процессоре.

Тестирование проводилось в локальной сети *Fast Ethernet* (*100 Мбит/с*), в качестве узлов которой были использованы компьютеры со следующими характеристиками: процессор *Intel Pentium 4* с тактовой частотой *2,66 ГГц*, *1 ГБ* оперативной памяти, жесткий диск объемом *160 ГБ*. Результаты тестирования приведены на рис. 2 и рис. 3.

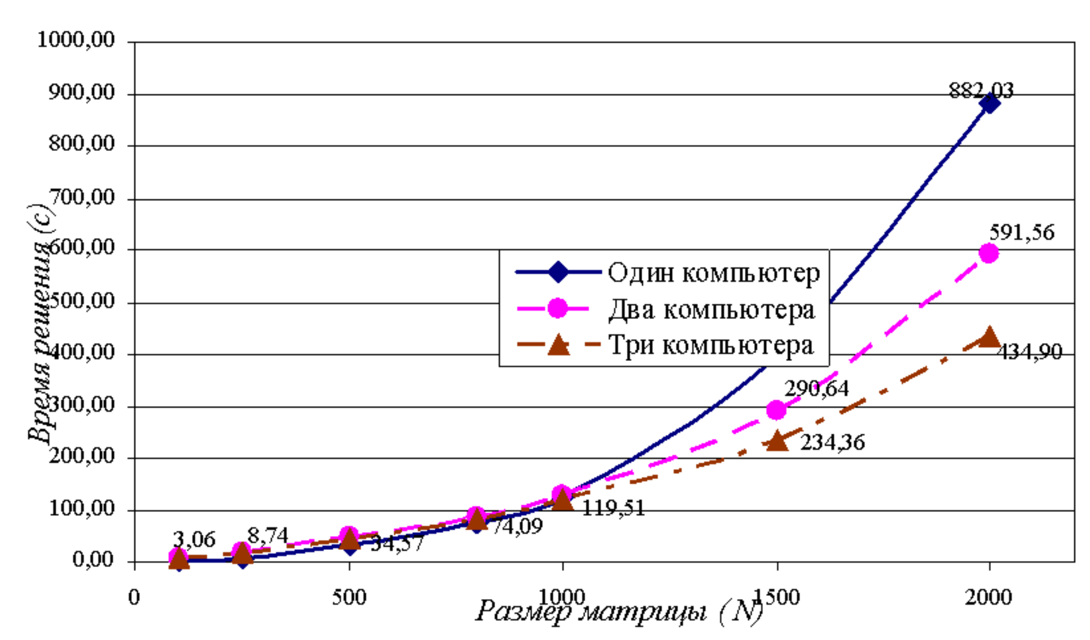


Рисунок 2 – Зависимость времени решения от числа уравнений для различного количества процессов

Как видно из графиков, при решении систем с небольшим количеством уравнений система менее эффективна. Это в первую очередь связано с латентностью *Fast Ethernet* и с использованием синхронизационных барьеров при решении.

При увеличении размерности матрицы сетевые задержки остаются такими же, а время расчета матрицы на каждом цикле алгоритма увеличивается, благодаря чему мы получаем большую загрузку задействованных в решении процессоров, чем достигается хорошее распараллеливание задачи.

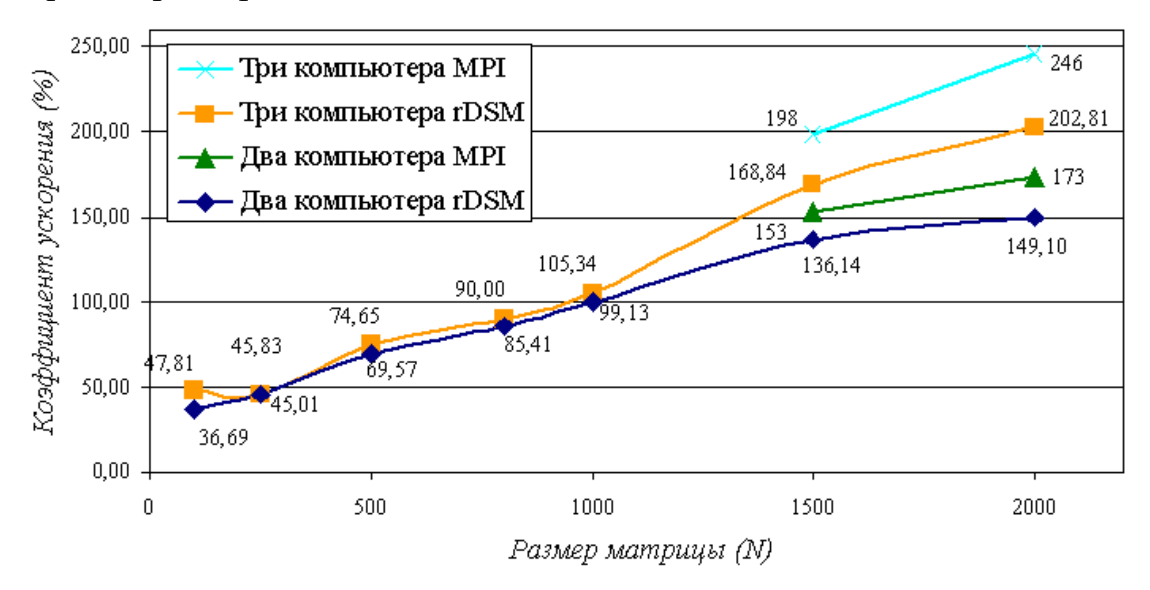


Рисунок 3 – Зависимость коэффициента ускорения решения от числа уравнений для различного количества процессов в rDSM и MPI

Сравнительный анализ с приложениями, разработанными с использованием коммуникационной библиотеки *MPI*, показывает, что rDSM и MPI практически сравнимы по производительности.

Заключение

В результате проведенных исследований предложена методика разработки распределенных приложений для кластерных систем, построенных на базе стандартных

сетевых технологий с улучшенными параметрами обработки данных. Она включает создание DSM-памяти на основе модели регионов и механизмы репликации объектов, обеспечивающих когерентность общей распределенной памяти.

Отличительная особенность проекта – интеграция различных типов памяти в рамках масштабируемого виртуального адресного пространства кластерной архитектуры, что позволит создавать эффективные инструментальные средства для проектирования сложных распределенных приложений. Предлагаемая программная реализация позволяет разрабатывать платформенно-независимые системы.

Внедрение разработанной методики обеспечивает интеграцию различных типов памяти в рамках масштабируемого виртуального адресного пространства кластерной архитектуры. Это дает возможность создавать эффективные инструментальные средства для проектирования сложных распределенных приложений, требующих больших объемов данных при решении задач в корпоративных GRID-сетях, построенных на основе стандартных высокоскоростных сетевых технологий.

Литература

1. Буза М.К. Параллельная обработка данных в модульных архитектурах / М.К. Буза // Вестник компьютерных и информационных технологий. – 2007. – № 6. – С. 51-56.
2. Буза М.К. Методы и средства распределенной обработки данных / М.К. Буза, Л.Ф. Зимянин // Выбраныя навуковыя працы Беларускага Дзяржаўнага Універсітэту : у 7 т. Т. 6: Матэматыка. – Мн. : БДУ, 2001. – С. 92-116.
3. Корнеев В.В. Параллельные вычислительные системы / Корнеев В.В. – М. : Нолидж, 1999. – 320 с.
4. Шпаковский Г.И. Программирование для многопроцессорных систем в стандарте MPI / Г.И. Шпаковский, Н.В. Серикова. – Мн. : БГУ, 2002. – 323 с.
5. Буза М.К. Архитектура компьютеров : учебник для университетов / Буза М.К. – Мн. : Изд-во «Новое знание», 2007. – 559 с.
6. Буза М.К. Программная поддержка распределенной общей памяти для кластерных архитектур / М.К. Буза, Л.Ф. Зимянин // Материалы V Международной конф.-форума (Минск, 16 – 17 ноября 2009). Ч. II. – С. 210-213.

Literatura

1. Bouza M.K. Vestnik komp’juternyh i informacionnyh tehnologij. № 6. 2007. S. 51-56.
2. Korneev V.V. Parallel’nye vychislitel’nye sistemy. M: Nolidzh. 1999. 320 s.
3. Shpakovskij G. I. Programmirovanie dlja mnogoprocessornyh i sistem v standarte MPI. Mn.: BGU. 2002. 323 s.
4. Bouza M.K. Vybranyja navukovyja pracy Belaruskaga Dzjarzhaŭnaga Universitjetu: U 7t. T. 6. Matjematyka. Mn.: BDU. 2001. S. 92-116.
5. Bouza M.K. Arhitektura komp’juterov. Uchebnik dlja universitetov. Mn. Izd-vo “Novoe znanie”. 2007. 559 s.
6. Bouza M.K. Materialy V Mezhdunarodnoj foruma konf. Minsk. 16-17 nojabrja 2009. Ch. II. S. 210-213.

RESUME

M.K. Bouza

Programming Support DSM-Memory

Programming system DSM – memory on the base of distributed objects is suggested. System supported data transfer in common address. Technology XSTM was used for replication objects between processors.

Статья поступила в редакцию 31.05.2012.